Umelá inteligencia

Zadanie č.2 - Zenová záhrada

(Zadanie 2a)

Autor: Marek Čederle
AIS ID: 121193 (xcederlem)
Cvičiaci: Ing. Nina Masaryková
(Streda 15:00)

Úvod

Ako spustiť program (Windows)

Je potrebné aby sme mali súbory main.cpp a garden.h v jednom adresáry. Otvoríme si príkazový riadok v adresári so súbormi a skompilujeme program pomocou kompilátora g++. Zadáme nasledujúce príkazy:
> g++ main.cpp -o main.exe
> .\main.exe

Po spustení program vypíše tieto inštrukcie pre pokračovanie:

Automatický vstup:

```
Enter 0 for automatic input or 1 for manual input: 0
If you wish to print min, max, average enter 1, else 0: 0
```

Manuálny vstup:

```
Enter 0 for automatic input or 1 for manual input: 1
If you wish to print min, max, average enter 1, else 0: 0

Enter the number of rows in the matrix: 10
Enter the number of columns in the matrix: 12
Enter number of generations (recommended 128-256): 128
Enter number of farmers in generation (MUST BE EVEN NUMBER, i.e. 128): 64
Enter number of stones (at least 1): 6
Indexing start from 0
Possible stone positions (from assignment):
1 5
2 1
3 4
4 2
6 8
6 9
Range: (0,0) (9,11)
Enter the position of stone (x y) number 1: 1 5
Enter the position of stone (x y) number 2: 2 1
Enter the position of stone (x y) number 3: 3 4
Enter the position of stone (x y) number 4: 4 2
Enter the position of stone (x y) number 5: 6 8
Enter the position of stone (x y) number 6: 6 9
```

Opis problému:

Máme záhradu, ktorú potrebujeme pohrabať. V záhrade sa môžu nachádzať prekážky napríklad kamene, ktoré musí farmár pri hrabaní obísť. Môže ísť však iba jedným smerom. Teda zvislo alebo vodorovne, nikdy nie šikmo. Vždy musí začať na okraji záhrady. Mimo okraju môže chodiť ako chce. Smer ktorým začne je určený jeho génmi. Ak narazí na prekážku alebo na už pohrabané miesto tak pomocou iných génov sa otočí ak má kam, ak sa nedá skúsi opačnú stranu. Ak sa nemá kde otočiť tak ukončí hrabanie. Úspešná hra je taká kde farmár pohrabe celú záhradu, prípadne maximálny počet políčok. Našou úlohou je nájsť také gény, ktoré umožnia farmárovi pohrabať celú záhradu ak je to možné.

Implementácia

Na implementáciu som použil jazyk C++ a nasledovné programy:

- Vývojové prostredie (IDE) – Clion
- Kompilátor – g++
- Grafy a tabuľky – Microsoft Excel
- Dokumentácia – Microsoft Word

Opis riešenia

Definícia záhrady:

```
class Garden{
public:

    int **matrix;
    int rows;
    int columns;
```

- Matrix – dynamicky alokované dvojrozmerné pole (matica pre záhradu)
- Rows – počet riadkov
- Columns – počet stĺpcov

Definícia génov:

```
class Gene{
public:
    vector<vector<unsigned int>> genes_vector;
    vector<int> turn_genes;

    int turn_genes_size;
    int max_size;
    unsigned int n, m;
```

- Genes_vector – 2D vektor všetkých pozičných génov jedného jedinca
- Turn_genes – vektor génov, ktoré sa použijú pri prekážkach
- Turn_genes_size – veľkosť následne vytvoreného vektora turn_genes
- Max_size – počet génov pre jedinca
- N – počet riadkov
- M – počet stĺpcov

Definícia farmára:

```
class Farmer{
public:
    Gene *genes;
    Garden *garden;
    int max_genes;
    int farmer_number;
    int fitness;
    int stones_count;
    int columns;
    int rows;
```

- Genes – objekt génov pre daného jedinca
- Garden – objekt záhrady pre daného jedinca
- Max_genes – počet génov pre jedinca
- Farmer_number – poradové číslo farmára v generácii
- Fitness – hodnota fitness pre jedinca
- Stones_count – počet kameňov v záhrade, použije sa pre veľkosť génov pre zatáčanie pri prekážkach
- Columns – počet stĺpcov
- Rows – počet riadkov

Definícia generácie:

```
class Generation{
public:
    int max_genes;
    int generation_number;
    int farmers_count;
    int matrix_rows;
    int matrix_columns;
    int turn_genes_count;

    vector<vector<int>> fitness_through_generations;
    vector<Farmer*> farmer;
```

- Max_genes – počet génov pre jedinca
- Generation_number – poradové číslo danej generácie
- Farmers_count – počet farmárov v generácii
- Matrix_rows – počet riadkov záhrady
- Matrix_columns – počet stĺpcov záhrady
- Turn_genes_count – počet génov pre zatáčanie
- Fitness_through_generations – 2D vektor kde sú uložené indexy farmára a jeho fitness pre jednu generáciu
- Farmer – vektor objektov typu farmár

Program pri spustení funguje nasledovne:

Vytvorí sa prvá generácia. Na jej vytvorenie sa použijú zadané údaje či už automaticky alebo manuálne a pridá sa do vektora generácií v main funkcii. Nasleduje vonkajší for cyklus ktorý bude iterovať do počtu zadaných generácií. Vnútorný for cyklus bude iterovať cez všetkých farmárov v danej generácii. Farmári v prvej generácii si náhodne vygenerujú gény funkciou randomGenerateGene(). Táto funkcia zabezpečí aby tieto gény boli unikátne a aby sa z nich dalo začať na okraji záhrady. Následne sa nastavia pozície kameňov v záhrade pre každého farmára. Potom farmár prejde (pohrabe) záhradu pomocou funkcie walkGarden().

```
for(int i=0; i < this->max_genes; i++)
{
    int localX = this->genes->genes_vector[i][0];
    int localY = this->genes->genes_vector[i][1];
    unsigned int hitCounter = 0;
    bool doneWalkedLine = false;
    int lastX = -1;
    int lastY = -1;


    // skip gene if farmer cant start walking from that position
    if(this->garden->matrix[localX][localY] != 0)
    {
       continue;
    }

    direction dir = NONE;

    // get direction from which side farmer should go straight
    if(localX == 0)
    {
        dir = DOWN;
    }
    else if (localX == this->garden->rows-1)
    {
        dir = UP;
    }
    else if (localY == 0)
    {
        dir = RIGHT;
    }
    else if (localY == this->garden->columns-1)
    {
        dir = LEFT;
    }
    else
    {
        // ERROR
        return;
    }
```

Táto funkcia (walkGarden) funguje nasledovne:

Nastavíme si počiatočné pozície vektorov a ďalšie pomocné premenné. Zistíme či sa vôbec dá začať na danej pozícii. Ak nie tak ideme na ďalší gén. Ak áno pokračujeme ďalej a zistíme ktorým smerom sa má farmár vydať. Potom nasleduje while cyklus ktorý zabezpečuje aby pohrabal jednu líniu záhrady. Najskôr zistí či už nie je na konci záhrady. Ak je, tak ukončí aktuálny gén a prejde sa na ďalší. Ak nie je pokračuje v hrabaní. Ak sa farmár zacyklí tak ukončíme hrabanie a prejdeme na ďalšieho farmára. Ak sa nezacyklí pokračujeme ďalej. Nasleduje switch ktorý rozhoduje ako budeme zapisovať pohyb.

Ukážka switchu pre jeden smer:

```
switch(dir)
{
    case RIGHT:
        // if farmer is on the end of walkable line, terminate the while loop
        if(localY+1 > this->garden->columns-1)
        {
            this->garden->matrix[localX][localY] = printCounter;
            doneWalkedLine = true;
            break;
        }
        // if farmer can go straight
        if(this->garden->matrix[localX][localY+1] == 0)
        {
            this->garden->matrix[localX][localY] = printCounter;
            ++localY;
        }
        // if farmer cant go straight, change direction
        else
        {
            direction temp_dir = changeDirection( current_dir: dir,  value: this->genes->turn_genes[hitCounter % this->genes->turn_genes_size]);
            if(canSwitchDir( dir: temp_dir, localX, localY))
            {
                dir = temp_dir;
            }
            else
            {
                temp_dir = oppositeDir( dir: temp_dir);
                if(canSwitchDir( dir: temp_dir, localX, localY))
                {
                    dir = temp_dir;
                }
                else
                {
                    return;
                }
            }

            hitCounter++;
            lastX = localX;
            lastY = localY;
        }
        break;
```

Ak sa v danom smere nedá ísť ďalej tak ukončíme hrabanie línie podľa daného génu a ideme na ďalší. Ak sa dá ísť ďalej tak normálne pokračujeme inak sa treba otočiť. Pri otočení sa skúša že či sa dá ísť ďalej ak sa otočíme, ak nie skúsime opačnú stranu a ak sa nedá otočiť vôbec tak sme sa dostali do bodu z ktorého sa nedá vrátiť takže ukončíme hrabanie farmára. Toto isté sa deje pre každý smer.

Po pohrabaní (prípadnom nepohrabaní) zistíme hodnotu fitness pomocou funkcie fitnessFunction().

```
void fitnessFunction()
{
    int done = 0;

    for(int i=0; i<this->garden->rows; i++)
    {
        for(int j=0; j< this->garden->columns; j++)
        {
            if(this->garden->matrix[i][j] != 0 && this->garden->matrix[i][j] != -1)
            {
                done++;
            }
        }
    }

    this->fitness = done;
}
```

Následne pridáme index farmára v generácii a jeho fitness hodnotu do nášho vektora pre fitness hodnoty v jednej generácii.

Následne testujeme že či sa nám podarilo nájsť riešenie. Ak áno vypíšeme v ktorej generácii a ktorý farmár je riešením a taktiež vypíšeme jeho gény a prechody po záhrade. Následne všetko dealokujeme. Ak nie tak pokračujeme ďalším farmárom.

Na konci jednej generácie podľa nastavenia si môžeme vypísať minimálne, maximálne a priemerné hodnoty fitness počas generácie.

Na konci vnútorného for cyklu usporiadame vektor, kde sú indexy farmárov a ich fitness hodnoty.

Ak sa nám nepodarí nájsť riešenie v prvej generácii tak ideme na ďalšiu generáciu.

Pri ďalších generáciách s využitím výberu, kríženia a mutovania bude vyzerať nasledovne.

Pre každú jednu ďalšiu generáciu sa náhodne v daný moment rozhodne ktorý spôsob výberu jedincov sa použije.

Výber génov pomocou rulety:

```cpp
vector<unsigned int> rouletteSelection(vector<vector<int>> fitness_and_farmer)
{
    unsigned seed = std::chrono::system_clock::now().time_since_epoch().count();
    srand(seed);

    unsigned int sum = 0;
    vector<unsigned int> fitness_and_farmer_out;
    vector<vector<int>> temp_in = fitness_and_farmer;


    for(int i= 0; i < temp_in.size(); i++)
    {
        sum += temp_in[i][1];
    }

    // Roulette wheel selection
    for(int i = 0; i < fitness_and_farmer.size() / 2; i++)
    {
        int random_num = rand() % sum;
        int current_sum = 0;

        // Find the selected individual based on the random number
        for (int j = 0; j < temp_in.size(); j++)
        {
            current_sum += temp_in[j][1];
            if (random_num < current_sum)
            {
                fitness_and_farmer_out.push_back(temp_in[j][0]);
                sum = sum - temp_in[j][1];

                temp_in.erase( position: temp_in.begin() + j);
                break;
            }
        }
    }


    return fitness_and_farmer_out;
}
```

Do rulety sa ako parameter pošle vektor kde sú indexy farmárov a ich fitness hodnoty s tým že sú už usporiadané zostupne. Zistíme sumu všetkých fitness hodnôt. Následne náhodne vyberieme jedinca s tým že čím vyššia fitness tak tým má väčšiu šancu na výber. Toto opakujeme kým nemáme dostatok vybraných jedincov. Vyberáme iba polovičný počet farmárov ktorý sa potom budú krížiť aby sme mali plný počet.

Výber génov pomocou hodnoty fitness:

```
vector<unsigned int> fitnessOrder(vector<vector<int>> fitness_and_farmer)
{
    vector<unsigned int> fitness_and_farmer_out;

    for(int i=0; i < fitness_and_farmer.size() / 2; i++)
    {
        fitness_and_farmer_out.push_back(fitness_and_farmer[i][0]);

    }

    return fitness_and_farmer_out;
}
```

Do tejto funkcie sa tiež ako parameter pošle vektor kde sú indexy farmárov a ich fitness hodnoty s tým že sú už usporiadané zostupne. Vyberieme iba najlepšiu polovicu.

Po výbere nasleduje kríženie jedincov:

Prvú polovicu krížime tak že zoberieme prvú polovicu génov z k-teho rodiča a druhú polovicu génov z k+1 – teho rodiča.
Druhú polovicu krížime tak že zoberieme prvú polovicu génov z k-teho rodiča a druhú polovicu génov z k+2 – teho rodiča.

Gény pre zatáčanie skopírujeme z prvého rodiča.

Následne zavoláme funkciu na mutovanie.

```
void mutateGene()
{
    unsigned seed = std::chrono::system_clock::now().time_since_epoch().count();
    srand(seed);

    for(int i = 0; i < this->genes->max_size; i++)
    {
        unsigned int probability = rand() % 100;

        // comparing variable means how many % chance there is for mutation
        if(probability < 25)
        {
            unsigned int geneToMutate = rand() % this->genes->max_size;
            unsigned int x;
            unsigned int y;

            do
            {
                x = rand() % this->genes->n;
                y = rand() % this->genes->m;
            }
            while(x != 0 && y != 0 && x != this->genes->n-1 && y != this->genes->m-1);

            this->genes->genes_vector[geneToMutate][0] = x;
            this->genes->genes_vector[geneToMutate][1] = y;
        }
    }
}
```

Táto funkcia zabezpečí že pre každý gén je 25% pravdepodobnosť že zmutuje.

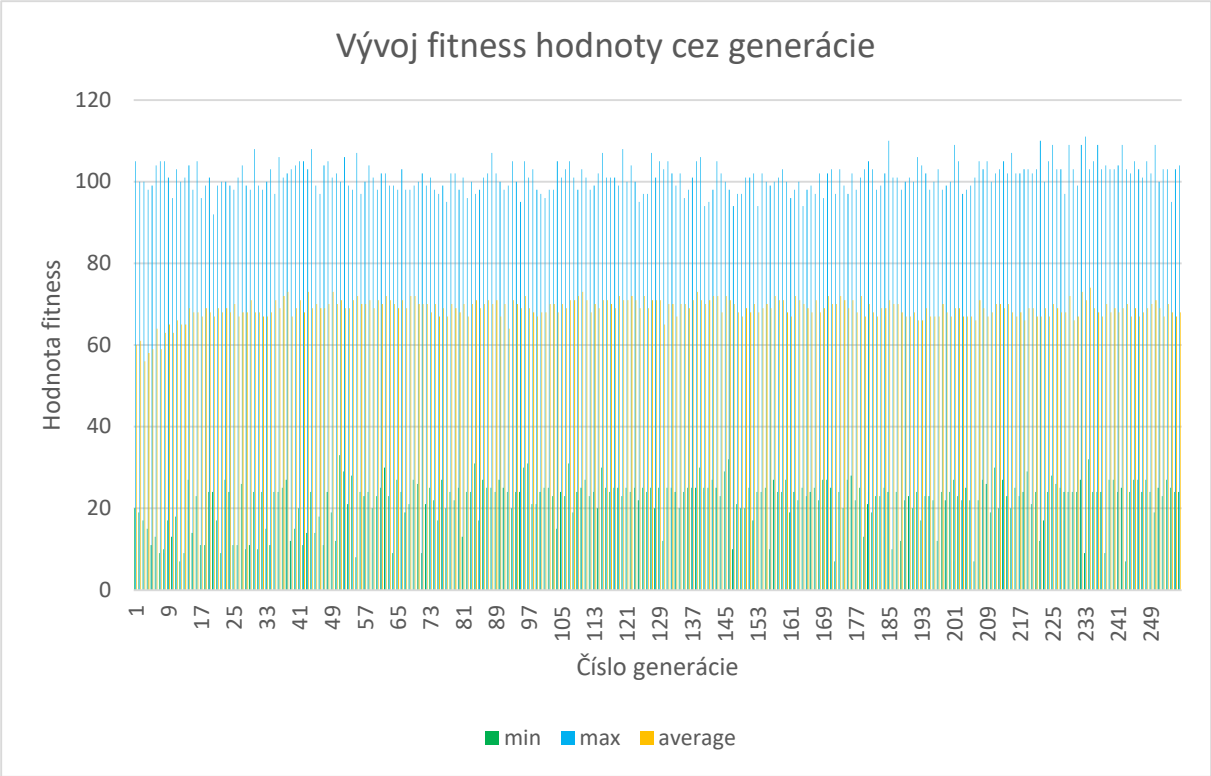
Ak sa riešenie nenájde ani v tejto generácii tak sa vytvorí nová rovnakým spôsobom a tak ďalej kým nenájdeme riešenie alebo vyčerpáme generácie.

Testovanie a záver

Implementáciu som testoval primárne na nastaveniach kameňov aké boli na obrázku v zadaní. Moja implementácia vie nájsť riešenie ale nie vždy aj keď je to možné. Je to z dôvodu zlej optimalizácie môjho riešenia. Jedno zo zlepšení by bolo iný spôsob mutovania prípadne ohodnocovanie jednotlivých génov farmára, nie iba fitness celého farmára.

Tabuľka a graf, ktorý reprezentujú vývoj fitness hodnoty jedincov cez generácie:

generation number	min	max	average
0	20	105	60
1	19	100	61
2	17	100	56
3	15	98	58
4	11	99	59
5	13	104	64
6	9	105	59
7	10	105	63
8	17	101	65
9	13	96	63
10	18	103	66
11	7	100	65
12	9	101	65
13	27	104	69
14	14	98	68
15	23	105	68
16	11	96	67
17	11	99	69
18	24	101	68
19	24	92	67
20	17	99	69
21	9	100	68
22	27	100	69
23	24	99	68
24	11	98	70
25	11	101	67
26	26	104	68
27	10	99	68
28	11	98	71
29	24	108	68
30	10	99	68
31	24	98	67
...	...	...	...



Z grafu je vidieť že na začiatku sa nám fitness zväčšuje a potom keďže je veľa farmárov a generácií tak sa nám fitness ustáli s prípadnými výkyvmi kvôli mutáciám.

Príklad nájdeného riešenia:

```
Solution:
Generation: 226 Farmer: 94
Genes:
{0,8} {9,6} {9,0} {0,8} {9,11} {0,9} {9,7} {5,0} {9,9} {7,0} {3,11} {0,7} {0,5} {2,11} {9,4} {9,5} {9,1} {9,3} {0,5} {9,6} {0,6} {9,8}
Turn genes:
{-1} {1} {-1} {1} {-1} {1}
 3  9  9  9  9  9  2  6  1  5  8 10
 3  9  9  9  9  S  2  6  1  5  8 10
 3  S 11 11  9  9  2  6  1  5  8 10
 3 11 11 11  S  9  2  6  1  5  8  8
 3 11  S 11 11  9  2  6  1  5  5  5
 3 11 12 12 11  9  2  6  1  1  1  1
 3 11 12 12 11  9  2  6  S  S  4  4
 3 11 12 12 11  9  2  6  7  7  4  4
 3 11 12 12 11  9  2  6  7  7  4  4
 3 11 12 12 11  9  2  6  7  7  4  4
```

Príklad takmer nájdeného riešenia:

```
Generation: 188 Farmer: 98 Fitness: 113
Genes:
{2,0} {0,7} {9,2} {8,0} {9,11} {0,3} {0,2} {9,3} {0,9} {9,3} {1,11} {0,10} {0,0} {4,11} {0,0} {7,11} {9,9} {0,9} {9,5} {4,11} {9,9} {0,9}
Turn genes:
{1} {-1} {-1} {-1} {-1} {-1}
10 10  6  5  5  0  3  2  8  8  9  4
10 10  6  5  5  S  3  2  8  8  9  4
 1  S  6  5  5  5  3  2  8  8  9  4
 1  6  6  5  S  5  3  2  8  8  9  4
 1  6  S  5  5  5  3  2  8  8  9  4
 1  6  3  3  3  3  3  2  8  8  9  4
 1  6  3  7  7  7  7  2  S  S  9  4
 1  6  3  7 12 12  7  2 11 11  9  4
 1  6  3  7 12 12  7  2 11 11  9  4
 1  6  3  7 12 12  7  2 11 11  9  4
```

Zdroje

http://www2.fiit.stuba.sk/~kapustik/zen.html
Prednášky z predmetu UI (Umelá inteligencia)